Model Railway Animation: Part 2 Expanded, Input Controls
By David King

In this expanded text of the original article published in The Canadian we will continue by calibrating the photo resistors to work like switches that can be used to activate the grade crossing flashers.

Calibrating the Photo Resistors

Calibrating is important as the lighting conditions that your photo resistors need to work in can vary greatly on your layout. If your layout is in a fixed location and you only operate in a day mode the lighting will most likely not change during your time on the layout so calibrating once when you first power up your layout should be sufficient. If your lighting conditions on your layout changes while operating you would need to re-calibrate the photo resistors.

The reason we want to calibrate the photo resistors is so that we know what the normal level of light on the sensors will be so that we can tell when the light is being blocked. This blocked value will be the value we are looking for to trip our railway grade flashing lights.

The calibration of the sensors is done in the void setup portion of the sketch since we usually only need to run this once. To start with the calibration will use a **while()**. This while function will be true for a time of 2 seconds and we can use the function **millis()** to read how much time has passed in milliseconds since the last power up or reset of the Uno. Since we are doing this calibration when powering up the Uno or just after resetting the Uno we can take advantage of an internal clock located in the main microprocessor chip on our Uno.

In the **while()** we will read the current value from the photo resistors and we will save the highest value we see. Remember that when light is on the sensors the value from the photo resistors will be low. Once the **while()** has finished we will add 200 to the low value and save this as the high value. The maximum rage for values from the photo resistors is from 0 to 1023 so using 200 as the difference between an un-blocked and a block sensor should be enough of a difference to prevent getting false high value triggers.

The only other items in the void setup are lines of coding to let us use the serial monitor so that we can tell what the sensor is using as the low and high values from our calibration.

Here is my code and I would suspect that yours will be similar but it doesn't need to be exactly the same. You can also use the last wiring configuration that we used for the previous sketch in the main article.

Photoeye_Calibration

```
// Photoeye_Calibration.ino by David King
// This is a sketch used to teach you how to calibrate the photo
// resistors.

// include any associated files in this location

// declare any variables needed in you file here
const int photoEye = A0;    // These variables are used for all of the photoeye needs
int photoEyeValue;
int photoEyeLow = 0;
int photoEyeHigh;

int ledPhotoEyeDark = 13;   // Sets pin for the on-board led

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);                     // Used to troubleshoot the file
  pinMode(ledPhotoEyeDark, OUTPUT);       // Set pin as an output

  while (millis() < 2000)                 // Calibrate the photo resistors
  {
    photoEyeValue = analogRead(photoEye);
    if (photoEyeValue > photoEyeLow)
    {
      photoEyeLow = photoEyeValue;        // Set the Low value of the photoeye
    }
  }
  photoEyeHigh = photoEyeLow + 200;       // Set the High value of the photoeye

  Serial.print("Calibration of Photoeye, Low = "); // Display the calibration results
  Serial.print(photoEyeLow);
  Serial.print(", High = ");
  Serial.println(photoEyeHigh);
}

```

Once we figure out the calibration of the photo resistor we should add some additional code to see the operation using a LED. Keeping it simple is the key here so we can use the LED to let us know each time that one or both of the photo resistors have enough light blocked. This would be similar to railcars passing over the photo resistors as the cars are moving along the tracks. Add the following code to test this.

```
void loop() {
  // put your main code here, to run repeatedly:

  photoEyeValue = analogRead(photoEye);     // Obtain the current value from the photoeye

  if (photoEyeValue > photoEyeHigh)         // Check for the photoeye to be unlit
  {
    digitalWrite(ledPhotoEyeDark, HIGH);    // Enable on-board led while photoeye is unlit
  } else
  {
    digitalWrite(ledPhotoEyeDark, LOW);     // Disable on-board led when photoeye is lit
  }
  Serial.print("photoEyeValue = ");
  Serial.println(photoEyeValue);
}
```

Now that we have the photo resistors working we should use them to trigger the grade crossing flasher lights. This can be done with very little change to the existing sketch. We will need to add a little coding in the void loop and we also need to add the new variable declarations and set the pinMode() for the additional LEDs. Here are the additions and new void loop code.

Add this code to the declarations.

```
int ledFlasher1 = 10;      // Sets pin for the 1st grade crossing LED
int ledFlasher2 = 11;      // Sets pin for the 2nd grade crossing LED
boolean turnOnFlasher = false;  // Use this to enable the grade crossing LEDs
int timeDelay = 250;       // Sets the timing of the flasher LEDs
```

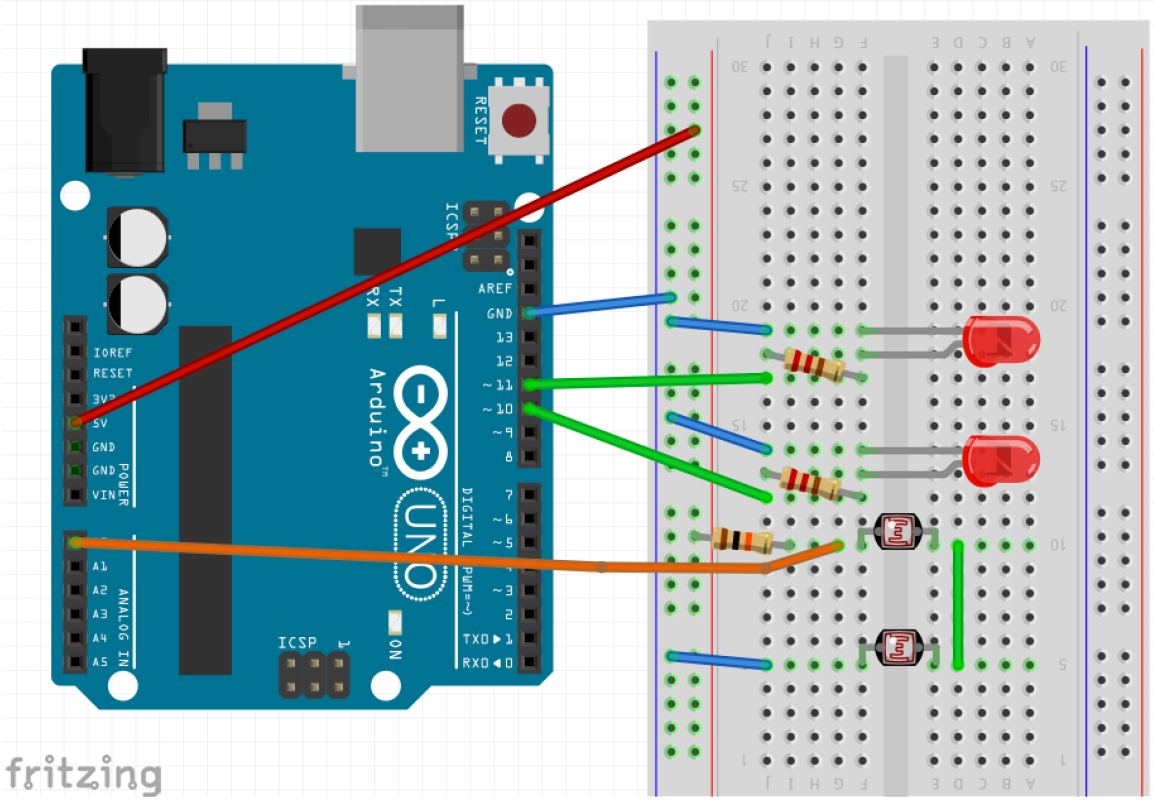
Add this code to the void setup.

```
pinMode(ledFlasher1, OUTPUT);        // Set pin as an output
pinMode(ledFlasher2, OUTPUT);        // Set pin as an output
```

Update the void loop code.

```
void loop() {
  // put your main code here, to run repeatedly:

  photoEyeValue = analogRead(photoEye);     // Obtain the current value from the photoeye

  if (photoEyeValue > photoEyeHigh)         // Check for the photoeye to be unlit
  {
    digitalWrite(ledPhotoEyeDark, HIGH);    // Enable on-board led while photoeye is unlit
    turnOnFlasher = true;
  } else
  {
    digitalWrite(ledPhotoEyeDark, LOW);     // Disable on-board led when photoeye is lit
    turnOnFlasher = false;
  }
  Serial.print("photoEyeValue = ");
  Serial.println(photoEyeValue);

  if(turnOnFlasher == true)
  {
    digitalWrite(ledFlasher1, HIGH);        // Cycle through the flasher LEDs
    digitalWrite(ledFlasher2, LOW);
    delay(timeDelay);

    digitalWrite(ledFlasher1, LOW);
    digitalWrite(ledFlasher2, HIGH);
    delay(timeDelay);
  }

  if(turnOnFlasher == false)
  {
    digitalWrite(ledFlasher1, LOW);         // Turn off the flasher LEDs
    digitalWrite(ledFlasher2, LOW);
  }
}
```

Here is the updated wiring required.



Now that the grade crossing flasher is being triggered by the lack of light on the photo resistors when should add an off delay to the trigger. What I mean by that is that we should continue to operate the grade flashing lights for a few seconds after light has been restored to all of the photo resistors. Right now each time the photo resistors have full light restored the grade flasher lights turn off very quickly. This causes a jerky or unsteady operation of the grade flashing lights as cars move from blocking the light to allowing the light to reach the photo resistors. To smoothen out the operation we will add a 6 second off delay for the trigger. To do this we will use a timer to maintain the trigger for the 6 seconds once the light is restored to the photo resistors. If the light is blocked prior to the 6 seconds finishing the timing operation the timer is reset and the 6 second timer starts over next time the photo resistors receive unblocked light.

A Timer

If we do a little searching we will find that there is no timer function but with some code we can create a timer. To do this we need to determine what steps will need to be taken to accomplish this function and learn a new data type to use the internal timer (clock) inside of the Uno. The new data type is *unsigned long* which starts at 0 and ends at 4,294,967,295 or (2^{32}-1). We will use this data type to read the current time in the Uno which is in milliseconds using the **millis()**. The code required would

be *timeCurrent = millis();* and this needs to be added in the void loop section. I'll show the completed code a little later on line 55.

Next we need to set a goal time for resetting the off delay timer. To do this we simply add 6000 (6 seconds) to the timeCurrent every time we see that at least one of the photo resistors has its light block. Luck for us we already check for this event in our sketch so we need to add the line of code, *timeGoal = timeCurrent + offDelayTime*; to the true condition of one of the **if(photoEyeValue > photoEyeHigh)** shown on line 55. Inside of the true condition we set the true state of the Boolean value *turnOnFlasher* on line 61 and this will now be used for not only starting the grade flasher LEDs but also for the time check for the off delay timer. At this time we will also remove the code located in the false condition of this same **if()** used for setting the *turnOnFlasher* state to false.

The last thing we need to do is set the state of *turnOnFlasher* to false when the *timeGoal* is less than the *timeCurrent* value. This is accomplished by using an **if()** starting on line 69.

Remember to add the declarations of the new value names as shown in the completed code below. No changes are needed to the wiring of the LEDs or photo resistors for this to work.

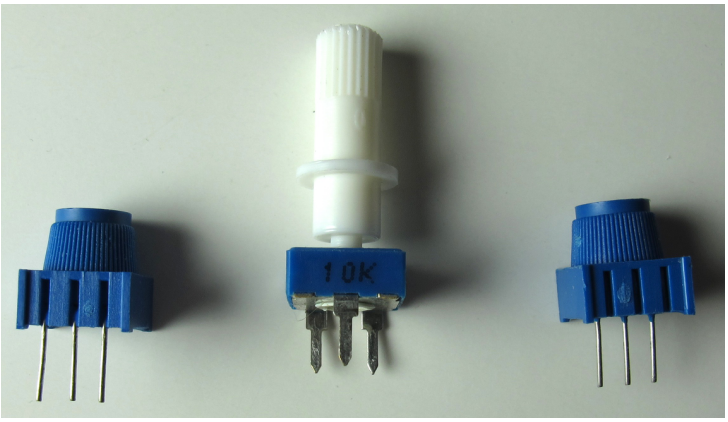
Photoeye_Grade_Crossing_2

```
1  // Photoeye_Grade_Crossing_2.ino by David King
2  // This is a sketch used to teach you how to calibrate the photo
3  // resistors and add the grade crossing flasher LEDs.
4  // An off delay timer has been added to maintain the grade flasher LEDs for
5  // a period of 6 seconds after the light has been restored to the photo
6  // resistors.
7
8  // include any associated files in this location
9
10 // declare any variables needed in you file here
11 const int photoEye = A0;    // These variables are used for all of the photoeye needs
12 int photoEyeValue;
13 int photoEyeLow = 0;
14 int photoEyeHigh;
15
16 int ledPhotoEyeDark = 13;   // Sets pin for the on-board led
17
18 int ledFlasher1 = 10;       // Sets pin for the 1st grade crossing LED
19 int ledFlasher2 = 11;       // Sets pin for the 2nd grade crossing LED
20 boolean turnOnFlasher = false;  // Use this to enable the grade crossing LEDs
21 int timeDelay = 250;        // Sets the timing of the flasher LEDs
22
23 unsigned long timeCurrent;  // Used for storage of the surrent time
24 unsigned long timeGoal;     // Used for storage of timer de-activation time
25 int offDelayTime = 6000;    // Used to set the off delay time, 6 seconds
26
```

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);                     // Used to troubleshoot the file
  pinMode(ledPhotoEyeDark, OUTPUT);       // Set pin as an output
  pinMode(ledFlasher1, OUTPUT);           // Set pin as an output
  pinMode(ledFlasher2, OUTPUT);           // Set pin as an output

  while (millis() < 2000)                 // Calibrate the photo resistors
  {
    photoEyeValue = analogRead(photoEye);
    if (photoEyeValue > photoEyeLow)
    {
      photoEyeLow = photoEyeValue;        // Set the Low value of the photoeye
    }
  }
  photoEyeHigh = photoEyeLow + 200;       // Set the High value of the photoeye

  Serial.print("Calibration of Photoeye, Low = "); // Display the calibration results
  Serial.print(photoEyeLow);
  Serial.print(", High = ");
  Serial.println(photoEyeHigh);
}

void loop() {
  // put your main code here, to run repeatedly:

  photoEyeValue = analogRead(photoEye);     // Obtain the current value from the photoeye

  timeCurrent = millis();                   // Obtain the current time from the Uno

  if (photoEyeValue > photoEyeHigh)         // Check for the photoeye to be unlit
  {
    timeGoal = timeCurrent + offDelayTime;  // Set the reset time for the off delay timer
    digitalWrite(ledPhotoEyeDark, HIGH);    // Enable on-board led while photoeye is unlit
    turnOnFlasher = true;                   // Enable the off delay timer
  } else
  {
    digitalWrite(ledPhotoEyeDark, LOW);     // Disable on-board led when photoeye is lit
  }
  Serial.print("photoEyeValue = ");
  Serial.println(photoEyeValue);

  if (timeGoal < timeCurrent)
  {
    turnOnFlasher = false;                  // Reset the off delay timer
  }

  if(turnOnFlasher == true)
  {
    digitalWrite(ledFlasher1, HIGH);        // Cycle through the flasher LEDs
    digitalWrite(ledFlasher2, LOW);
    delay(timeDelay);

    digitalWrite(ledFlasher1, LOW);
    digitalWrite(ledFlasher2, HIGH);
    delay(timeDelay);
  }

  if(turnOnFlasher == false)
  {
    digitalWrite(ledFlasher1, LOW);         // Turn off the flasher LEDs
    digitalWrite(ledFlasher2, LOW);
  }
}
```

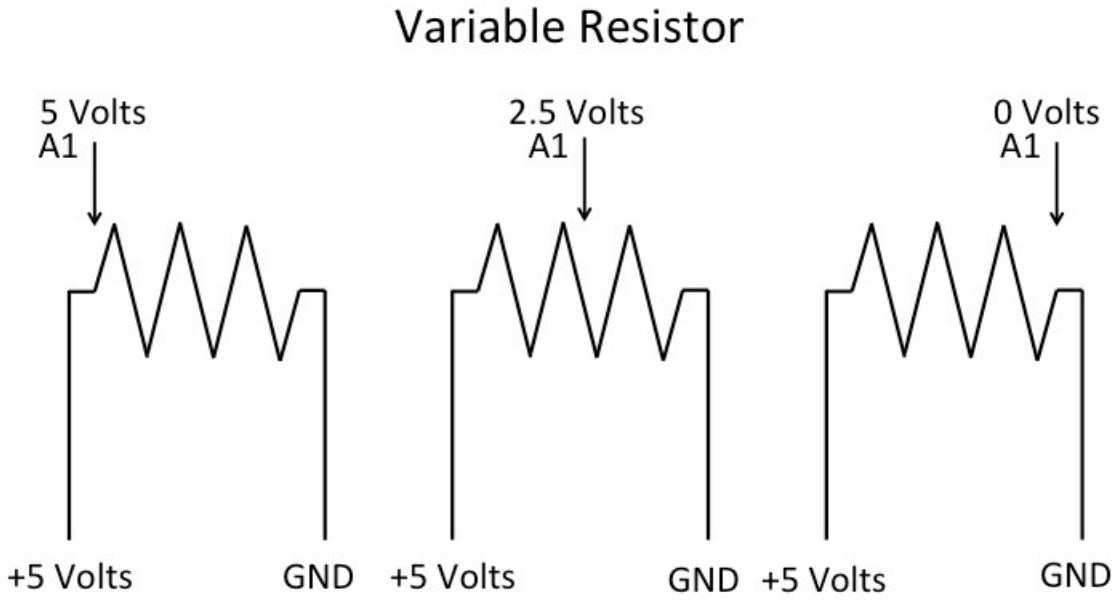
One more addition

This may look like a lot but we have one more little step to improve this grade crossing flasher LEDs and that would be to have the ability to vary the rate of flash. To do this we are going to add another external component, the potentiometer or variable resistor. The connections required for this device are very simple as it has only 3 connection points. The outer 2 connections are connected to +5 volts and ground, one connection to each. The middle connection is connected to A1 another of the analog connections on the Uno.



These 3 variable resistors are from Adafruit, Arduino and SparkFun, left to right.

Electrically all 3 of these devices are identical, 10,000 ohms each, but they vary physically. To adjust the resistance on the middle pin as compared to the outside pins you only need to turn the knob. This will vary the voltage on the middle pin when the outside pins are connected to +5 volts and ground. Below is a drawing that may help explain this.



The A1 pin as an analog input can read this varying of the voltage from the potentiometer. The Uno converts this voltage signal to a digital value that ranges from 0 to 1023. This is considered a 10-bit input (2^{10}). We could use this value as our timeDelay variable but we may want to manipulate this number as the delay for the flashers should never be 0 milliseconds and 1.023 seconds would be way too long. A range somewhere in between these two extremes would be better. Yes we could just make sure that we don't adjust the potentiometer too low or too high but to get the most our of the variable resistor we could just rescale the value to 50ms to 750ms. Luck for use that there is a built-in function to do just this.

The **map(x, a, b, c, d)** is simple to use so let me explain how it works. First the **x** is where we will insert the value from the A1 pin. Next the **a** is the minimum value of the input , **b** is the maximum value of the input. Finally **c** is the new minimum value and **d** is the new maximum value that will be used for the timeDelay variable. The completed function would look like this.

```
delayTime = map(potInput, 0, 1023, 50, 750);
```

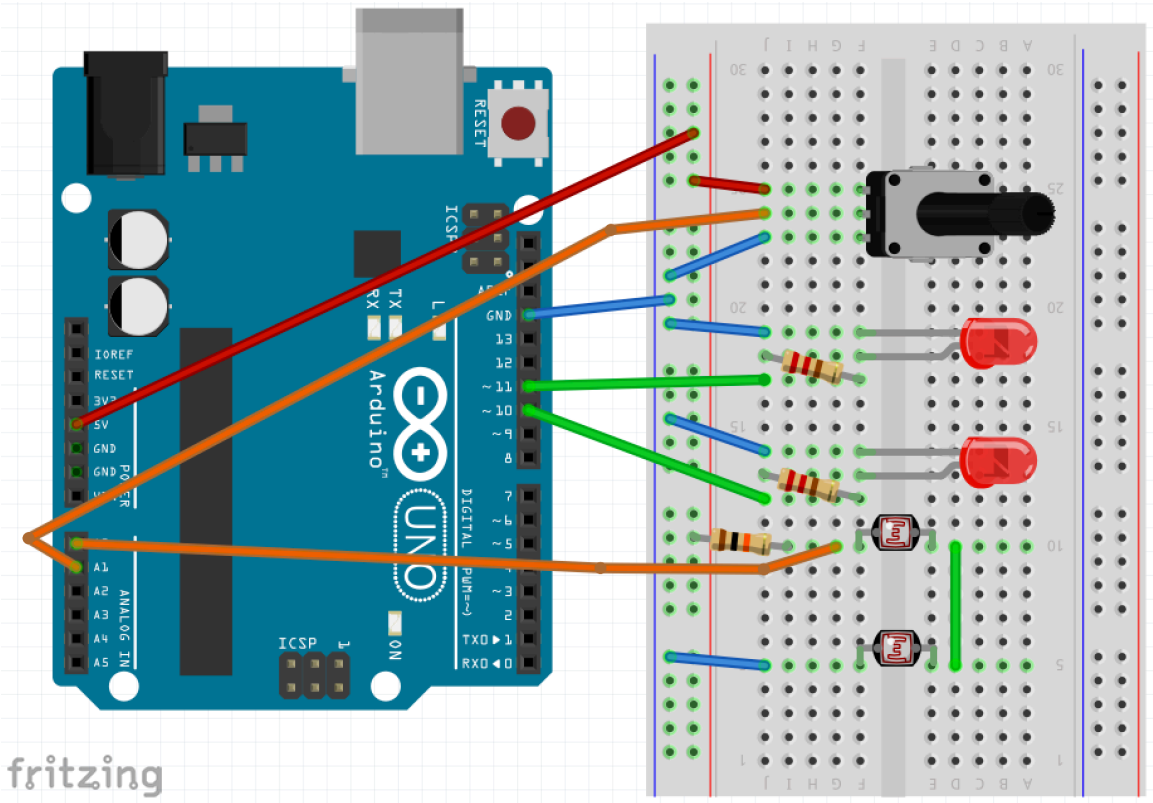
The new variable potInput is the raw input value on pin A1.

The completed code is shown after the wiring diagram below. Here is a list of the code changes made and their current rung numbers.

Rung 7, added additional comment for the variable resistor.
Rungs 23 & 24, added the new variable that need to be declared.
Rung 58, read the analog input from pin A1.
Rung 59, map the input from pin A1 to a range of 50ms to 750ms.

If you follow the wiring diagram and the code changes you should now have a fully operational grade crossing flasher with an adjustable flash rate. Good Luck.

Here is wiring diagram for the addition on variable resistor.



Photoeye_Grade_Crossing_3

```
1  // Photoeye_Grade_Crossing_3.ino by David King
2  // This is a sketch used to teach you how to calibrate the photo
3  // resistors and add the grade crossing flasher LEDs.
4  // An off delay timer has been added to maintain the grade flasher LEDs for
5  // a period of 6 seconds after the light has been restored to the photo
6  // resistors.
7  // A variable resistor has been added to make the flash rate adjustable.
8
9  // include any associated files in this location
10
11 // declare any variables needed in you file here
12 const int photoEye = A0;    // These variables are used for all of the photoeye needs
13 int photoEyeValue;
14 int photoEyeLow = 0;
15 int photoEyeHigh;
16
17 int ledPhotoEyeDark = 13;   // Sets pin for the on-board led
18
19 int ledFlasher1 = 10;       // Sets pin for the 1st grade crossing LED
20 int ledFlasher2 = 11;       // Sets pin for the 2nd grade crossing LED
21 boolean turnOnFlasher = false;  // Use this to enable the grade crossing LEDs
22 int timeDelay = 250;        // Sets the timing of the flasher LEDs
23 const int potInput = A1;    // This sets the input pin for the variable resistor
24 int potValue;               // This set a variable for reading the variable resistor
25
26 unsigned long timeCurrent;  // Used for storage of the surrent time
27 unsigned long timeGoal;     // Used for storage of timer de-activation time
28 int offDelayTime = 6000;    // Used to set the off delay time, 6 seconds
29
```

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);                     // Used to troubleshoot the file
  pinMode(ledPhotoEyeDark, OUTPUT);       // Set pin as an output
  pinMode(ledFlasher1, OUTPUT);           // Set pin as an output
  pinMode(ledFlasher2, OUTPUT);           // Set pin as an output

  while (millis() < 2000)                 // Calibrate the photo resistors
  {
    photoEyeValue = analogRead(photoEye);
    if (photoEyeValue > photoEyeLow)
    {
      photoEyeLow = photoEyeValue;        // Set the Low value of the photoeye
    }
  }
  photoEyeHigh = photoEyeLow + 200;       // Set the High value of the photoeye

  Serial.print("Calibration of Photoeye, Low = "); // Display the calibration results
  Serial.print(photoEyeLow);
  Serial.print(", High = ");
  Serial.println(photoEyeHigh);
}

```

```
void loop() {
  // put your main code here, to run repeatedly:

  photoEyeValue = analogRead(photoEye);     // Obtain the current value from the photoeye

  potValue = analogRead(potInput);              // Obtains the value from the variable resistor
  timeDelay = map(potValue, 0, 1023, 50, 750);  // Converts to a time range of 50ms to 750ms

  timeCurrent = millis();                   // Obtain the current time from the Uno

  if (photoEyeValue > photoEyeHigh)         // Check for the photoeye to be unlit
  {
    timeGoal = timeCurrent + offDelayTime;  // Set the reset time for the off delay timer
    digitalWrite(ledPhotoEyeDark, HIGH);    // Enable on-board led while photoeye is unlit
    turnOnFlasher = true;                   // Enable the off delay timer
  } else
  {
    digitalWrite(ledPhotoEyeDark, LOW);     // Disable on-board led when photoeye is lit
  }
  Serial.print("photoEyeValue = ");
  Serial.println(photoEyeValue);

  if (timeGoal < timeCurrent)
  {
    turnOnFlasher = false;                  // Reset the off delay timer
  }

```

```
  if(turnOnFlasher == true)
  {
    digitalWrite(ledFlasher1, HIGH);        // Cycle through the flasher LEDs
    digitalWrite(ledFlasher2, LOW);
    delay(timeDelay);

    digitalWrite(ledFlasher1, LOW);
    digitalWrite(ledFlasher2, HIGH);
    delay(timeDelay);
  }

  if(turnOnFlasher == false)
  {
    digitalWrite(ledFlasher1, LOW);         // Turn off the flasher LEDs
    digitalWrite(ledFlasher2, LOW);
  }
}
```

Conclusion

This has concluded this lesson but be sure to join us in the next issue of *The Canadian* as I this series of articles continue with a focus of making things move. Also send me any comments or ideas for future projects to David King, directordavid@caorm.org